Midterm Project

Anhelina Bondarenko

Std: 220104004928

**Expected grade: BA**

**Program Report: AdaBank System**

File hierarchy :

project/

├── bin/ (not exist before compilation)

│ ├── client

│ └── server

├── src/

│ ├── bank.h

│ ├── server.c

│ ├── client.c

│ └── teller.c

├── test/

│ ├── client01.file

│ ├── client02.file

│ ├── client03.file

│ └── client04.file

├── Makefile

├── AdaBank.bankLog

**Commands:**

**./bin/server AdaBank /tmp/bank\_server.fifo**

./bin/client test/client01.file /tmp/bank\_server.fifo

./bin/client test/client02.file /tmp/bank\_server.fifo

./bin/client test/client03.file /tmp/bank\_server.fifo

**1. Overview of IPC and Synchronization Mechanisms**

The AdaBank system employs a combination of inter-process communication (IPC) and synchronization techniques to manage concurrent client requests and ensure data consistency. The design revolves around a client-server architecture where the server processes transactions via *Teller* processes, while clients communicate through FIFO (named pipe) channels. Below are the key mechanisms:

**Inter-Process Communication (IPC):**

* **FIFOs (Named Pipes):**
  + Clients send requests to the server via the server FIFO *(/tmp/bank\_server.fifo).*
  + Each client creates a unique FIFO (e.g., */tmp/client\_<PID>*) to receive responses.
  + The server reads requests in batches from its FIFO and dispatches them to Teller processes.
  + Responses are written back to the client’s FIFO atomically using a semaphore-guarded write operation.
* **Shared Memory:**
  + A shared memory segment (/*bank\_shm*) stores the bank’s account data (SharedData structure), including account IDs, balances, and client counts.
  + This allows all Teller processes (child processes of the server) to access and modify the same data structure concurrently.

**Synchronization:**

* **Semaphores:**
  + *sem****(****/bank\_sem****)*:** Ensures mutual exclusion when accessing the SharedData structure. Critical sections (e.g., account creation, balance updates) are guarded by this semaphore.
  + *fifo\_mutex****(****/bank\_fifo\_mutex****)*:** Protects write operations to client FIFOs to prevent interleaved messages.
  + *req\_sem****(****/bank\_req\_sem****)*:** Coordinates batch processing of client requests. The server waits on this semaphore until clients signal new requests.
* **Process Management:**
  + Each client request is handled by a dedicated *Teller* process (via fork()), isolating transaction logic and preventing blocking.
  + The server uses waitpid() to ensure Tellers complete processing before accepting new batches.

**2. Core Function Logic**

*handle\_client(Request \*req)*

**Purpose:** Validates client requests, assigns client/account IDs, and sends initial responses.  
**Workflow:**

1. **Semaphore Acquisition:** The function begins by acquiring sem to safely access SharedData.
2. **Account Validation:**
   * For new clients (account\_id = "NEW"), increments client\_count and assigns a sequential client ID.
   * For existing accounts, searches SharedData.accounts for a matching ID. If absent, treats it as a new client.
3. **Response Generation:** Constructs a confirmation message (e.g., *"Client01 connected..deposit 300 credits"*).
4. **FIFO Write:** Uses *fifo\_mutex* to atomically write the response to the client’s FIFO.

**Critical Sections:**

* Modifications to *client\_count* and accounts are protected by sem.
* FIFO writes are guarded by *fifo\_mutex* to ensure atomicity.

*deposit(void \*arg)*

**Purpose:** Processes deposit requests, updates account balances, and logs transactions.  
**Workflow:**

1. **Semaphore Acquisition:** Locks *sem* to enter the critical section.
2. **New Account Handling:**
   * If *account\_id is* "NEW", creates a new account with a unique BankID\_XX and initializes its balance.
   * Rejects the request if MAX\_ACCOUNTS is reached.
3. **Existing Account Handling:**
   * Searches for the account ID in SharedData.accounts.
   * Updates the balance and logs the transaction via *write\_log().*
4. **Response Dispatch:** Sends a success/failure message to the client’s FIFO using *fifo\_mutex*.

**Edge Cases:**

* Handles maximum account limits.
* Validates account existence before modifying balances.

*withdraw(void \*arg)*

**Purpose:** Processes withdrawals, validates balances, and removes empty accounts.  
**Workflow:**

1. **Semaphore Acquisition:** Locks sem to enter the critical section.
2. **Account Validation:**
   * Searches for the account ID in SharedData.accounts.
   * Checks if the balance is sufficient for the withdrawal.
3. **Balance Adjustment:**
   * Deducts the amount and updates the log.
   * If the balance reaches zero, removes the account from SharedData.accounts using memmove().
4. **Response Dispatch:** Notifies the client of success/failure via FIFO.

**Edge Cases:**

* Rejects withdrawals from invalid or insufficient accounts.
* Handles account removal atomically to prevent data corruption.

**3. Signal Handling and Cleanup**

The server and clients gracefully handle termination signals (e.g., SIGINT):

* **Server Cleanup (***cleanup()***):**
  + Unmaps shared memory (munmap()) and unlinks IPC resources (shm\_unlink, sem\_unlink).
  + Removes the server FIFO and updates the log file.
* **Client Cleanup:**
  + Closes FIFOs and releases semaphores to prevent resource leaks.

**4. Transaction Logging**

The *write\_log()* function appends transaction details to AdaBank.bankLog with timestamps. Each entry includes:

* Account ID, transaction type (D/W), amount, balance, and timestamp.

**Test Case Description: AdaBank Transaction Processing**

**Server-Side Execution**

1. **Initialization**:
   * The server (AdaBank) starts, initializes a new database (no prior logs), and listens on /tmp/bank\_server.fifo.
   * Shared memory and semaphores are set up to manage concurrent access to account data.
2. **Batch Processing**:
   * **Batch 1 (4 Clients)**:
     + **Client01**: Deposits 300 credits, creating BankID\_01.
     + **Client02**: Withdrawal fails (invalid operation, e.g., insufficient balance or invalid account).
     + **Client03**: Deposits 1000 credits, creating BankID\_02.
     + **Client04**: Withdrawal fails.
   * **Batch 2 (2 Clients)**:
     + **Client01**: Withdraws 300 credits, closing BankID\_01.
     + **Client05**: Deposits 20 credits, creating BankID\_03.
   * **Batch 3 (5 Clients)**:
     + **Client03**: Withdraws 30 credits (balance: 970), deposits 200 credits (balance: 1170), withdraws 300 credits (balance: 870).
     + **Client06**: Deposits 2000 credits, creating BankID\_04.
     + **Client07**: Withdrawal fails.
3. **Termination**:
   * On receiving SIGINT (^C), the server closes active tellers, removes FIFOs/semaphores, updates logs, and exits gracefully.

Зображення, що містить текст, знімок екрана, Шрифт

Вміст, створений ШІ, може бути неправильним.

Зображення, що містить текст, знімок екрана, Шрифт, програмне забезпечення

Вміст, створений ШІ, може бути неправильним.

**Client-Side Execution**

**Client 01 (**test/client01.file**)**

* **Transactions**:
  + **Deposit 300**: Success → Account BankID\_01 created.
  + **Withdraw 200**: Fails (invalid operation).
  + **Deposit 1000**: Success → Account BankID\_02 created.
  + **Withdraw 275**: Fails.
* **Outcome**: Two accounts created; two withdrawals rejected.

**Client 02 (**test/client02.file**)**

* **Transactions**:
  + **Withdraw 300**: Success → BankID\_01 closed (balance zero).
  + **Deposit 20**: Success → Account BankID\_03 created.
* **Outcome**: One account closed, one new account created.

**Client 03 (**test/client03.file**)**

* **Transactions**:
  + **Withdraw 30**: Success → Balance updated to 970.
  + **Deposit 200**: Success → Balance updated to 1170.
  + **Withdraw 300**: Success → Balance updated to 870.
  + **Deposit 2000**: Success → Account BankID\_04 created.
  + **Withdraw 20**: Fails (likely invalid account).
* **Outcome**: Multiple balance updates on BankID\_02; one new account created.

Зображення, що містить текст, знімок екрана

Вміст, створений ШІ, може бути неправильним.

**Key Observations**

1. **Concurrency & Synchronization**:
   * Teller processes (e.g., PIDI$0137) handle requests in parallel.
   * Semaphores ensure atomic access to shared data (e.g., BankID\_01 modifications).
2. **FIFO Communication**:
   * Clients use unique FIFOs to receive responses.
   * Server writes responses atomically using fifo\_mutex to prevent interleaving.
3. **Error Handling**:
   * Invalid withdrawals are rejected with clear error messages (e.g., "Invalid operation").
   * Closed accounts are removed from the shared database.
4. **Logging**:

* All transactions are logged with timestamps and balances in server, client and AdaBank.bankLog

**Conclusion**

The AdaBank system effectively leverages FIFOs, shared memory, and semaphores to achieve concurrency and data consistency. Critical functions like handle\_client, deposit, and withdraw ensure thread-safe operations through meticulous semaphore management. While the design addresses core banking operations, future enhancements could focus on optimizing IPC performance and error recovery.